Machine Learning Project Report

Group 70

Authors:

Francisco Tavares (103402) francisco.carreira.tavares@tecnico.ulisboa.pt

Rodrigo Sereno (102785) rodrigo.sereno@tecnico.ulisboa.pt

1 Objectives

This report is grounded in two main subjects: regression and image analysis. For each, we will apply a range of tests and robust algorithms to ensure the study's accuracy and reliability. In the regression analysis section, we aim to tackle a linear regression problem that includes outliers and subsequently develop an Auto Regressive Exogenous (ARX) model. The image analysis component will focus on accurately classifying images of the Martian surface as cratered or non-cratered. Finally, we will apply image segmentation techniques to the images to delineate crater regions in the classified images.

2 Multiple Linear Regression with Outliers

In this task, our goal is to develop a multiple linear regression model that predicts a single dependent variable based on five independent variables. Approximately 25% of the dependent variable data points are impacted by human error, introducing outliers into the dataset. To address this, we implemented an outlier removal technique. Additionally, we explored various models to identify the best fit for the data, evaluating each models performance using the sum of squared errors (SSE). This approach allows us to assess and select the model that best captures the underlying relationships in the data.

2.1 Outlier Removal

To address the issue, we implemented a Python function that is designed to iteratively detect and remove data points with the largest prediction errors.

The function operates in an iterative loop, where it fits the model to the data, calculates predictions, and then computes absolute prediction errors. In each iteration, the function identifies the data point with the highest error and removes it from both X and Y. This process repeats until the specified number of outliers has been removed. The function then returns the cleaned datasets X_{cleaned} and Y_{cleaned}.

The decision to remove 48 outliers was done by an initial analysis using the RANSAC algorithm, which provided an estimate of the number of significant outliers in the dataset.

Figure 1 illustrates the total sum of errors across the iterations of outlier removal. In this instance, we removed 51 outliers to demonstrate that the error does not significantly decrease beyond this point. Although it is possible to remove more than 48 outliers and achieve a Total Sum of Absolute Errors lower than 7.51, we opted against this to avoid eliminating important data points. Removing too many outliers could lead to overfitting, making the model too well-suited to the specific dataset and potentially less effective when applied to new, unseen data points.

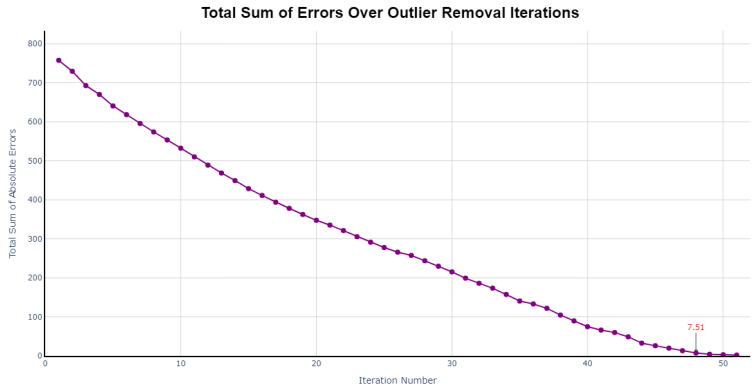

Figure 1: Total Sum of Errors Over Outlier Removal Iterations.

2.2 Models Used and Cross Validation

In our analysis, we employed four regression models: Linear Regression, Ridge Regression, Lasso Regression, and RANSAC Regression. RANSAC regressor was not covered in our lectures, it operates as an algorithm that iteratively fits a model to a subset of the data, thereby minimizing the influence of outliers. RANSAC randomly selects data points to construct a candidate model, evaluating how well it fits the majority of data points and discarding those that do not conform to the model. This process is repeated, and the model that best represents the inlier points is chosen as the final model.

To evaluate the performance of each model, we used K-Fold Cross-Validation with $k = 10$ folds. This technique allowed us to train and validate each model multiple times on different data subsets, enabling a more robust assessment of their predictive accuracy. For the Ridge and Lasso models, we applied hyperparameter tuning using RidgeCV and LassoCV, respectively. These methods automatically searched across a range of α values to find the optimal regularization strength, minimizing overfitting and improving model generalization. The ideal values of α for Ridge and Lasso were $\alpha = 0.00064$ and $\alpha = 0.0001$ respectively.

The performance of each model was measured using Sum of Squared Errors (SSE), Mean Squared Error (MSE), Median Absolute Deviation (MAD), and the R^2 score, providing a comprehensive comparison of their effectiveness in handling a dataset with outliers.

2.3 Results Discussion

For the four models and with 48 outliers removed the results that we obtained were the following:

Table 1: Model Performance Metrics

Model	SSE	MSE	MAD	R^2
Linear	1.241206	0.007984	0.018163	0.995122
Ridge Regression	1.252636	0.008061	0.019245	0.995073
Lasso Regression	1.241810	0.007987	0.018467	0.995122
RANSAC	1.241206	0.007984	0.018163	0.995122

Our results indicate that the best performance was achieved using both standard linear regression and the RANSAC regressor. It is noteworthy that both models yielded identical results. This similarity led us to conclude that, because we remove the outliers before running the RANSAC regressor, it doesn't find significant outliers to remove so it works as a normal linear regression.

We believe that if we had removed 50 outliers instead of 48 we would get better results in the submission. With this adjustment, using the RANSAC regressor would have resulted in a sum of squared errors (SSE) of 0.040343. However, we still believe that selecting a lower number of outliers contributes to the reliability of our implementation.

3 The ARX Model

In this task, we implement an ARX model to represent discrete-time input-output dynamical systems. The ARX model is defined by a linear difference equation, where the output $y(k)$ depends on its past values, the current and past values of the input $u(k)$, and a noise term $e(k)$.

We have a dataset of 2550 samples, split into a training set of 2040 samples and a test set of 510 samples. Our goal is to predict the last 400 values of the test output using the test input data, excluding the first 110 values to allow the model to stabilize. We will experiment with different values for the model order parameters n, m, and d to identify the best-performing configuration.

3.1 Time Series Split Cross Validation

When building and evaluating the ARX model, time series cross-validation (TSCV) is crucial due to the dataset's dynamic nature, where the order of samples matters. Unlike standard K-fold cross-validation, which shuffles data, TSCV maintains the sequence, ensuring the training set always precedes the testing set. This method prevents data leakage, where future information could incorrectly influence model performance. By using TSCV, we create a training and validation process that more accurately reflects real-world conditions, where predictions rely on past data.

1. **Initial Split**: Divide the dataset into k contiguous folds, preserving the order of observations.
2. **Training and Testing**: Train on earlier data and test on later data to simulate real-world predictions.
3. **Iterative Process**: Repeat k times, moving the test set forward while maintaining the training set's order, thus minimizing data leakage and providing a realistic model assessment.

In this analysis, we manually tuned the hyperparameters for the Ridge and Lasso models. Running RidgeCV and LassoCV within the Time Series Split validation, coupled with the three nested loops for testing different values of n, m, and d, resulted in excessively long runtimes. This manual approach allowed for more efficient experimentation without compromising the reliability of the results.

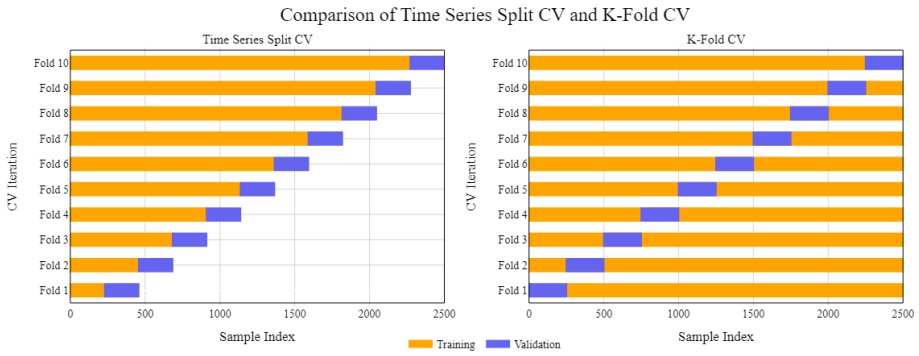

Figure 2: Visual representation of Time Series Split CV and K-Folds CV

3.2 Results Discussion

After running the code, the model parameters that achieved the best score were $m = 9$, $n = 9$ and $d = 6$, which resulted in the following average SSE after validation using the Time Series Split CV and the output predictions.

Model	SSE
Linear Regression	5.21595
Ridge Regression	5.21650
Lasso Regression	10.69556

Table 2: Sum of Squared Errors (SSE) for each model.

We achieved the best results with the normal linear regression model, which received the highest evaluation score. Unfortunately, during the submission process, we experienced a lapse and submitted the incorrect vector, leading to our disqualification. However, we had the opportunity to assess our model's performance, resulting in a sum of squared errors (SSE) of 6.8385. This performance would have secured us a rank of 49.

Figure 3 illustrates one of the validation folds, where the predicted outputs are plotted against the actual outputs. The plot clearly shows that our model performed really well in making predictions. Additionally, it highlights the reasoning behind the requirement to submit only the last 400 data points of the predicted test vector; because the model requires some time to "warm up" before it begins to generate accurate predictions.

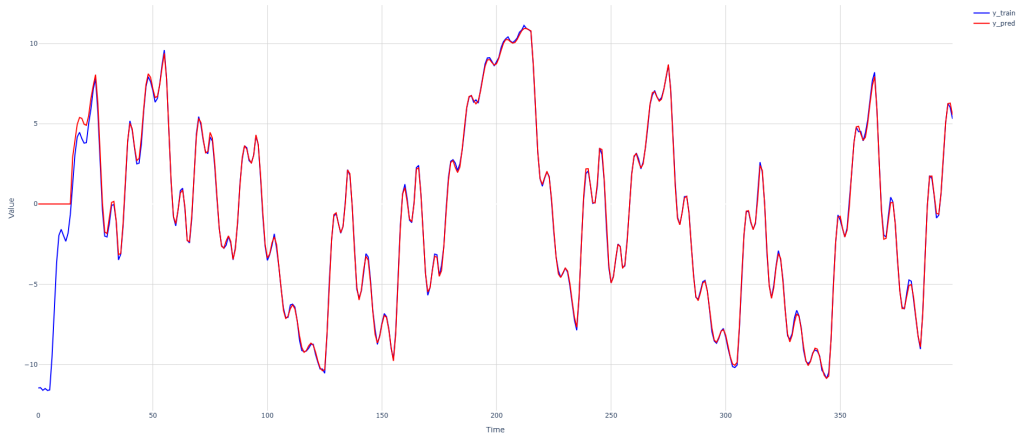

Figure 3: Predicted output of the validation fold plotted over the True output from the same fold.

4 Image classification

For this task we trained two different models to identify whether there were crater regions in 48x48 images or not and classify them accordingly, where images that appeared to have craters where classified as "1", and images without craters as "0". The two models chosen by our group where a Convolutional Neural Network (CNN) based approach and a Suport Vector Machine (SVM) approach, witch will be explained later.

4.1 Models Used & Architecture

As mentioned earlier there were used two models, an CNN and an SVM. The CNN computed is composed by eleven layers. Firstly, an input layer that receives the 48x48 images, followed by three pairs of Convolutional-Pooling Layers. Each Convolutional Layer is made up by 32, 64 and 128 3x3 kernels, respectively, where to each is applied the activation function "ReLU", and each Pooling Layer has a 2x2 pool size. Next we added an Flatten Layer to convert the 2D feature maps into 1D feature vectors, one Dense/Fully Connected Layer, where for the 128 neurons we applied the same activation function as in the Convolutional Layers, the "ReLU" function, and one Dropout Layer with a dropout rate of 50%. This means that half of its input is randomly dropped during the training, as a regularization technique to help prevent overfitting. Finally we add an Dense/Fully Connected Layer with two neurons where to each is applied the "sigmoid" activation function, that returns the model's output. The second model chosen was the Support Vector Machine (SVM), which was used to perform non-linear classification, using RBF as its kernel.

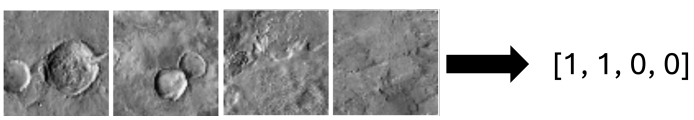

Figure 4: CNN/SVM

4.2 Hyperparameter Tuning & Validation

To validate our models, and prevent overfitting we used K-Folds Cross Validation. For the CNN model K-Fold CV was used with 10 splits, 20 epochs and a batching size of 32.

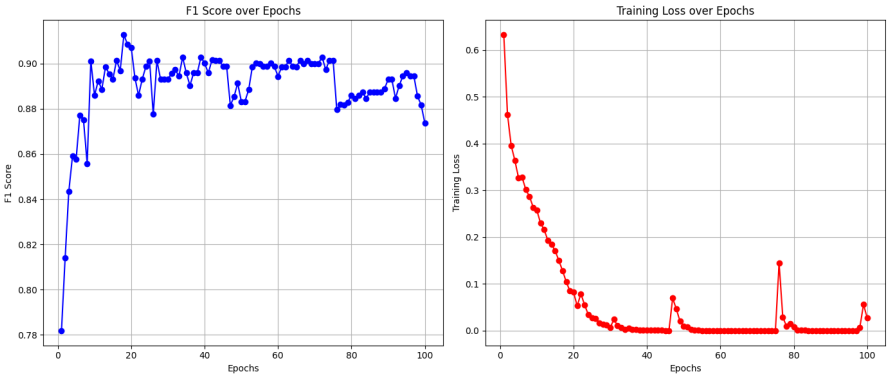

Figure 5: F1 Score and Training Loss over 100 Epochs

The plot above shows how the F1 Score and Training Loss change with the number of epochs for the CNN model without using cross validation. Looking at it we can see that after 20 epochs the change in the F1 score and Training Loss is not significant, and doesn't compensate the added computational cost It's important to denote that using a large number of epochs may cause overfit in our model, causing poor generalization to other datasets. To update the network weights was used the Adam optimizer with a learning rate of 0.001. These values were the ones who delivered the highest F1 Score. The values used in the parameters of the Layers and the optimizer, but also the number of pairs Convolutional-Pooling Layers were also chosen via experimentation, the F1 score obtained for the values chosen were the highest between the pool of values. To tune the parameters of the SVM, C and gamma, we used the function GridSearchCV(), that performs a grid search with cross validation to find the best combination for parameters gamma and C. The combinations of gamma and C values are tested across 5 folds to evaluate which settings yield the best performance according to the F1 score.

4.3 Data Imbalance

A first look at the Dataset shown that the classes were imbalanced, with, of the 2783 images, 1777 being labeled as "1" (with crater) and 1006 as "0" (without crater). To solve this issue, we opted to augment the training dataset in each fold of the K-Split, adding extra images to the minority class, where the extra images come from 90, 180 or 270 degree rotations on randomly selected images from the minority class. The data balance is only made in the training fold, to avoid the possibility of training and validating the model on the same images.

4.4 Extra Data

To further train the model, the extra training dataset was used in the *semi_supervised_learning*() function. The model makes two prediction rounds on the extra dataset and only saves predictions where the model is confident in both rounds. There is a confidence level threshold defined

by us (85%), and if the models confidence for that prediction is higher than the threshold in both rounds, and the prediction is the same in both predictions, it is saved. Those predictions are then added to the original training dataset, to better train our models.

4.5 Results

For both models with the optimal parameters and using the training Dataset the following F1 Scores were obtained:

Model	F1 Score
CNN	0.8930
SVM	0.7313

Table 3: F1 Score for each model.

The best results were achieved with the Convolutional Neural Network based approach, witch scored us an F1 Score of 0.9003 for the test Dataset.

5 Image segmentation

In the final task of this project, we trained two different models to identify crater regions in 48x48 images, with crater pixels labeled as '1' and the background as '0'. We used two distinct datasets, A and B: dataset A was used to train a Random Forest model, and dataset B was used to train a U-Net model, which will be explained in subsequent sections.

5.1 Data Imbalance

In this problem, the data imbalance was particularly pronounced, with 896,429 pixels labeled as '0' (background) and 363,859 pixels labeled as '1' (crater). To address this imbalance in dataset B, we applied class weighting, assigning a higher weight to the crater class ('1') during the training of the U-Net model. This weighting scheme allowed the model to focus more on detecting crater pixels, thereby mitigating the effect of the imbalance.

5.2 Models Used & Architecture

As mentioned, we trained a U-Net model using dataset B. The architecture consists of an encoder that progressively down-samples the input image through convolutional and pooling layers, capturing important contextual features. In parallel, the decoder upsamples these features to restore the original spatial dimensions of the image. A key feature of the U-Net is its skip connections, which link corresponding layers from the encoder to the decoder. These connections allow the model to preserve spatial information that may be lost during the down-sampling process, enhancing the accuracy of the segmentation results.

For the dataset A we trained a random forest which operates by constructing multiple decision trees during training, each built on a random subset of the data and features, thereby introducing diversity and reducing overfitting. In this process, the model learns to associate

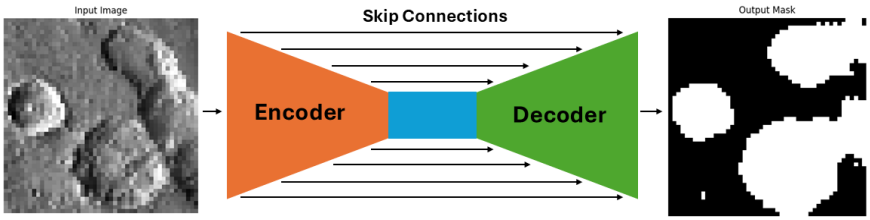

Figure 6: U-Net Architecture

the features of the patches with their respective labels by aggregating the predictions from all the decision trees through a majority voting mechanism.

5.3 Hyperparameter Tuning & Validation

Once again, for validation and to prevent overfitting in our models, we used K-Folds Cross Validation. To tune the hyperparameters of the U-Net, we experimented with various combinations of epochs and batch sizes, and we included dropout layers that randomly disable a fraction of neurons during training. This approach helps prevent overfitting by encouraging the model to learn more robust and diverse features, rather than relying on specific neurons. Additionally, we utilized the Adam optimizer with a learning rate of 0.001 to minimize the categorical cross-entropy loss function during training. For the Random Forest classifier, we explored different values for the minimum number of samples per leaf and the maximum depth of the trees to find the optimal configuration.

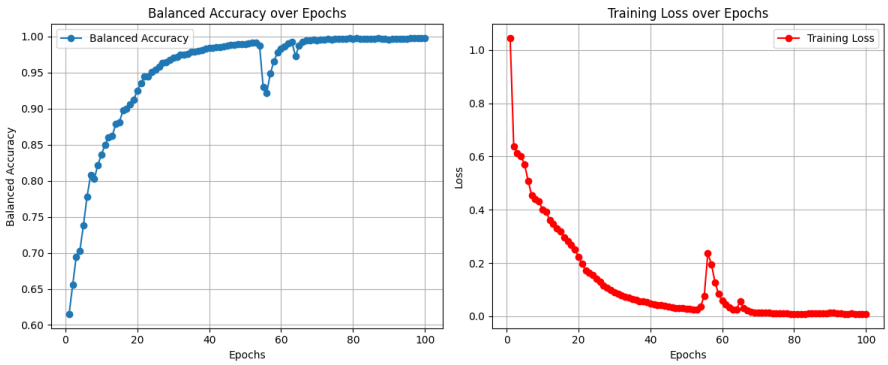

Figure 7: Balanced Accuracy and Training Loss over 100 Epochs

This plot shows the Balanced Accuracy (left) and Training Loss (right) of the model without using cross validation for over 100 training epochs. In the Balanced Accuracy plot, we observe a rapid increase within the first 25 epochs, reaching close to 97%. After this point, accuracy plateaus with minor fluctuations, indicating that the model has learned the core patterns necessary for good performance. In the Training Loss plot, the loss decreases significantly in the initial 25 epochs, then slowly stabilizes with only minor improvements, confirming that the model has converged.

Given these patterns, we decided to limit training to 25 epochs to avoid unnecessary computational costs. Additional epochs do not substantially improve performance, which implies that the model has already reached its optimal training capacity by this point.

The results from these experiments, following the data augmentation techniques that will be explained next, are as follows:

Modelo	Folds	Epochs	Batch Size	min_samples_leaf	max_depth	Balance Accuracy
Random Forest	5	-	-	1	10	0.63607
Random Forest	5	-	-	5	30	0.67157
Random Forest	5	-	-	10	50	0.67412
U-Net	5	25	128	-	-	0.70293
U-Net	5	15	128	-	-	0.71434
U-Net	5	15	64	-	-	0.74262
U-Net	5	15	16	-	-	0.79628
U-Net	5	25	16	-	-	0.80884

Figure 8: Results for different Hyperparameters

5.4 Data Augmentation

Finally, we decided to implement data augmentation to enhance the generalization of our model. To achieve this, we randomly rotated the images from the training folds by 90, 180, or 270 degrees. These augmented images were then concatenated with the original training data, effectively increasing the size and diversity of the training set. This approach allowed us to train the U-Net model on a more varied dataset, which is crucial for improving the models' robustness and performance in segmenting unseen data. Without data augmentation we got an average balanced score of 0.76468 across all 5 folds vs 0.80884 with data augmentation technique, so it improved our model accuracy.

6 Conclusion

In conclusion, the work we performed achieved excellent results across all tasks. Unfortunately, for the ARX model, we submitted an incorrect vector, which affected our grade. However, in the other tasks, we consistently achieved high classifications, and we were able to identify and explain areas for potential improvement. This project was valuable for applying class concepts and enabled us to explore beyond the curriculum. The additional research proved highly enriching, significantly deepening our understanding of machine learning.

References

[1] Scikit-learn: RANSAC Regressor: https://shorturl.at/AdmVw

[2] Scikit-learn: Time Series Split: https://shorturl.at/AoLvI

[3] Convolutional Neural Networks: https://shorturl.at/IRzb0

[4] Sckikit-learn: GridSearchCV https://shorturl.at/blqLu

[5] U-Net: https://shorturl.at/y3zM2